Sequential Protocol Composition in Maude-NPA[⋆]

Santiago Escobar[1], Catherine Meadows[2], José Meseguer[3], and Sonia Santiago[1]

[1] DSIC-ELP, Universidad Politécnica de Valencia, Spain
{sescobar,ssantiago}@dsic.upv.es
[2] Naval Research Laboratory, Washington, DC, USA
meadows@itd.nrl.navy.mil
[3] University of Illinois at Urbana-Champaign, USA
meseguer@cs.uiuc.edu

Abstract. Protocols do not work alone, but together, one protocol relying on another to provide needed services. Many of the problems in cryptographic protocols arise when such composition is done incorrectly or is not well understood. In this paper we discuss an extension to the Maude-NPA syntax and operational semantics to support dynamic sequential composition of protocols, so that protocols can be specified separately and composed when desired. This allows one to reason about many different compositions with minimal changes to the specification. Moreover, we show that, by a simple protocol transformation, we are able to analyze and verify this dynamic composition in the current Maude-NPA tool. We prove soundness and completeness of the protocol transformation with respect to the extended operational semantics, and illustrate our results on some examples.

1 Introduction

It is well known that many problems in the security of cryptographic protocols arise when the protocols are composed. Protocols that work correctly in one environment may fail when they are composed with new protocols in new environments, either because the properties they guarantee are not quite appropriate for the new environment, or because the composition itself is mishandled.

The importance of understanding composition has long been acknowledged, and there are a number of logical systems that support it. The Protocol Composition Logic (PCL) begun in [9] is probably the first protocol logic to approach composition in a systematic way. Logics such as the Protocol Derivation Logic (PDL) [4], and tools such as the Protocol Derivation Assistant (PDA) [1] and the Cryptographic Protocol Shape Analyzer (CPSA) [8] also support reasoning about composition. All of these are logical systems and tools that support

⋆ S. Escobar and S. Santiago have been partially supported by the EU (FEDER) and the Spanish MEC/MICINN under grant TIN 2007-68093-C02-02. J. Meseguer has been supported by NSF Grants CNS 07-16638 and CNS 09-04749.

© Springer-Verlag Berlin Heidelberg 2010

reasoning about the properties guaranteed by the protocols. One uses the logic to determine whether the properties guaranteed by the protocols are adequate. This is a natural way to approach composition, since one can use these tools to determine whether the properties guaranteed by one protocol are adequate for the needs of another protocol that relies upon it. Thus in [6] PCL and in [15] the authentication tests methodology underlying CPSA are used to analyze key exchange standards and electronic commerce protocols, respectively, via composition out of simpler components.

Less attention has been given to handling composition when model checking protocols. However, model checking can provide considerable insight into the way composition succeeds or fails. Often the desired properties of a composed protocol can be clearly stated, while the properties of the components may be less well understood. Using a model checker to experiment with different compositions and their results helps us to get a better idea of what the requirements on both the subprotocols and the compositions actually are.

The problem is in providing a specification and verification environment that supports composition. In general, it is tedious to hand-code compositions. This is especially the case when one protocol is composed with other protocols in several different ways. In this paper we propose a syntax and operational semantics for sequential protocol composition in Maude-NPA [10,11], a protocol specification and analysis tool based on unification and narrowing-based backwards search. Sequential composition, in which one or more child protocols make use of information obtained from running a parent protocol, is the most common use of composition in cryptographic protocols. We show that it is possible to incorporate it via a natural extension of the operational semantics of Maude-NPA. We have implemented this protocol composition semantics via a simple program transformation without any change to the tool. We prove the soundness and completeness of the transformation with respect to the semantics.

The rest of the paper is organized as follows. In Section 2 we introduce two protocol compositions that we will use as running examples: one example of one-parent-one-child composition, and another of one-parent-many-children composition. After some preliminaries in Section 3, in Section 4 we give an overview of the Maude-NPA tool and its operational semantics. In Section 5 we describe the new composition syntax and semantics. In Section 6 we describe the operational semantics of composition and the protocol transformation and give soundness and completeness results. In Section 7 we conclude the paper and discuss related and future work.

2 Two Motivating Examples

In both of our examples we build on the well-known Needham-Schroeder-Lowe (NSL) protocol [19]. The first example of protocol composition, which appeared in [16], is an example of a one-parent, one-child protocol, which is subject to an unexpected attack not noticed before. In this protocol, the participants use NSL to agree on a secret nonce. We reproduce the NSL protocol below.

1. $A \to B : \{N_A, A\}_{pk(B)}$
2. $B \to A : \{N_A, N_B, B\}_{pk(A)}$
3. $A \to B : \{N_B\}_{pk(B)}$

where $\{M\}_{pk(A)}$ means message M encrypted using the public key of principal with name A, N_A and N_B are nonces generated by the respective principals, and we use the comma as message concatenation.

The agreed nonce N_A is then used in a distance bounding protocol. This is a type of protocol, originally proposed by Desmedt [7] for smart cards, which has received new interest in recent years for its possible application in wireless environments [3]. The idea behind the protocol is that Bob uses the round trip time of a challenge-response protocol with Alice to compute an upper bound on her distance from him in the following protocol.

4. $B \to A : N'_B$
   Bob records the time at which he sent N'_B
5. $A \to B : N_A \oplus N'_B$
   Bob records the time he receives the response and checks the equivalence $N_A = N_A \oplus N'_B \oplus N'_B$. If it is equal, he uses the round-trip time of his challenge and response to estimate his distance from Alice.

where $\oplus$ is the exclusive-or operator satisfying associativity (i.e., $X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z$) and commutativity (i.e., $X \oplus Y = Y \oplus X$) plus the properties $X \oplus X = 0$ and $X \oplus 0 = X$. Note that Bob is the initiator and Alice is the responder of the distance bounding protocol, in contrast to the NSL protocol.

This protocol must satisfy two requirements. The first is that it must guarantee that $N_A \oplus N'_B$ was sent after N'_B was received, or Alice will be able to pretend that she is closer than she is. Note that if Alice and Bob do not agree on N_A beforehand, then Alice will be able to mount the following attack: $B \to A : N'_B$ and then $A \to B : N$. Of course, $N = N'_B \oplus X$ for some X. But Bob has no way of telling if Alice computed N using N'_B and X, or if she just sent a random N. Using NSL to agree on a $X = N_A$ in advance prevents this type of attack.

Bob also needs to know that the response comes from whom it is supposed to be from. In particular, an attacker should not be able to impersonate Alice. Using NSL to agree on N_A guarantees that only Alice and Bob can know N_A, so the attacker cannot impersonate Alice. However, it should also be the case that an attacker cannot pass off Alice's response as his own. But this is not the case for the NSL distance bounding protocol, which is subject to the following attack[1]:

a) Intruder I runs an instance of NSL with Alice as the initiator and I as the responder, obtaining a nonce N_A.
b) I then runs an instance of NSL with Bob with I as the initiator and Bob as the responder, using N_A as the initiator nonce.
c) $B \to I : N'_B$ where I does not respond, but Alice, seeing this, thinks it is for her.

[1] This is not meant as a denigration of [16], whose main focus is on timing models in strand spaces, not the design of distance bounding protocols.

d) $A \rightarrow I : N'_B \oplus N_A$ where Bob, seeing this thinks this is I's response.

If Alice is closer to Bob than I is, then I can use this attack to appear closer to Bob than he is. This attack is a textbook example of a composition failure. NSL has all the properties of a good key distribution protocol, but fails to provide all the guarantees that are needed by the distance bounding protocol. However, in this case we can fix the problem, not by changing NSL, but by changing the distance bounding protocol so that it provides a stronger guarantee:

4. $B \rightarrow A : \{N'_B\}$
5. $A \rightarrow B : \{h(N_A, A) \oplus N'_B\}$ where h is a collision-resistant hash function.

As we show in our analysis (not included in this paper but available online at `http://www.dsic.upv.es/~ssantiago/composition.html`), this prevents the attack. I cannot pass off Alice's nonce as his own because it is now bound to her name.

The distance bounding example is a case of a one parent, one child protocol composition. Each instance of the parent NSL protocol can have only one child distance bounding protocol, since the distance bounding protocol depends upon the assumption that N_A is known only by A and B. But because the distance bounding protocol reveals N_A, it cannot be used with the same N_A more than once.

Our next example is a one parent, many children composition, also using NSL. This type of composition arises, for example, in key distribution protocols in which the parent protocol is used to generate a master key, and the child protocol is used to generate a session key. In this case, one wants to be able to run an arbitrary number of child protocols.

In the distance bounding example the initiator of the distance bounding protocol was always the child of the responder of the NSL protocol and vice versa. In the key distribution example, the initiator of the session key protocol can be the child of either the initiator or responder of the NSL protocol. So, we have two possible child executions after NSL:

4. $A \rightarrow B : \{Sk_A\}_{h(N_A,N_B)}$	4. $B \rightarrow A : \{Sk_B\}_{h(N_A,N_B)}$
5. $B \rightarrow A : \{Sk_A; N'_B\}_{h(N_A,N_B)}$	5. $A \rightarrow B : \{Sk_B; N'_A\}_{h(N_A,N_B)}$
6. $A \rightarrow B : \{N'_B\}_{h(N_A,N_B)}$	6. $B \rightarrow A : \{N'_A\}_{h(N_A,N_B)}$

where Sk_A is the session key generated by principal A and h is again a collision-resistant hash function.

These two examples give a flavor for the variants of sequential composition that are used in constructing cryptographic protocols. A single parent instance can have either many children instances, or be constrained to only one. Likewise, parent roles can determine child roles, or child roles can be unconstrained. In this paper we will show how all these types of composition are specified and analyzed in Maude-NPA, using these examples as running illustrations.

3 Background on Term Rewriting

We follow the classical notation and terminology from [22] for term rewriting and from [20,21] for rewriting logic and order-sorted notions. We assume an *order-sorted signature* Σ with a finite poset of sorts $(\mathsf{S}, \leq)$ and a finite number of function symbols. We assume an S-sorted family $\mathcal{X} = \{\mathcal{X}_{\mathsf{s}}\}_{\mathsf{s} \in \mathsf{S}}$ of disjoint variable sets with each $\mathcal{X}_{\mathsf{s}}$ countably infinite. $\mathcal{T}_{\Sigma}(\mathcal{X})_{\mathsf{s}}$ denotes the set of terms of sort s, and $\mathcal{T}_{\Sigma,\mathsf{s}}$ the set of ground terms of sort s. We write $\mathcal{T}_{\Sigma}(\mathcal{X})$ and $\mathcal{T}_{\Sigma}$ for the corresponding term algebras. We write $\mathcal{V}ar(t)$ for the set of variables present in a term t. The set of positions of a term t is written $Pos(t)$, and the set of non-variable positions $Pos_{\Sigma}(t)$. The subterm of t at position p is $t|_p$, and $t[u]_p$ is the result of replacing $t|_p$ by u in t. A *substitution* σ is a sort-preserving mapping from a finite subset of $\mathcal{X}$ to $\mathcal{T}_{\Sigma}(\mathcal{X})$.

A *Σ-equation* is an unoriented pair $t = t'$, where $t \in \mathcal{T}_{\Sigma}(\mathcal{X})_{\mathsf{s}}$, $t' \in \mathcal{T}_{\Sigma}(\mathcal{X})_{\mathsf{s}'}$, and s and s' are sorts in the same connected component of the poset $(\mathsf{S}, \leq)$. For a set E of Σ-equations, an *E-unifier* for a Σ-equation $t = t'$ is a substitution σ s.t. $\sigma(t) =_E \sigma(t')$. A *complete* set of E-unifiers of an equation $t = t'$ is written $CSU_E(t = t')$. We say $CSU_E(t = t')$ is *finitary* if it contains a finite number of E-unifiers. A *rewrite rule* is an oriented pair $l \to r$, where $l \notin \mathcal{X}$ and $l, r \in \mathcal{T}_{\Sigma}(\mathcal{X})_{\mathsf{s}}$ for some sort $\mathsf{s} \in \mathsf{S}$. An *(unconditional) order-sorted rewrite theory* is a triple $\mathcal{R} = (\Sigma, E, R)$ with Σ an order-sorted signature, E a set of Σ-equations, and R a set of rewrite rules. The rewriting relation $\to_{R,E}$ on $\mathcal{T}_{\Sigma}(\mathcal{X})$ is $t \xrightarrow{p}_{R,E} t'$ (or $\to_{R,E}$) if $p \in Pos_{\Sigma}(t)$, $l \to r \in R$, $t|_p =_E \sigma(l)$, and $t' = t[\sigma(r)]_p$ for some σ. Assuming that E has a finitary and complete unification algorithm, the narrowing relation $\leadsto_{R,E}$ on $\mathcal{T}_{\Sigma}(\mathcal{X})$ is $t \overset{p}{\leadsto}_{\sigma,R,E} t'$ (or $\leadsto_{\sigma,R,E}$, $\leadsto_{R,E}$) if $p \in Pos_{\Sigma}(t)$, $l \to r \in R$, $\sigma \in CSU_E(t|_p = l)$, and $t' = \sigma(t[r]_p)$.

4 Maude-NPA's Execution Model

Given a protocol $\mathcal{P}$, a *state* in the protocol execution is a term t of sort State, $t \in T_{\Sigma_{\mathcal{P}}/E_{\mathcal{P}}}(X)_{\mathsf{State}}$, where $\Sigma_{\mathcal{P}}$ is the signature defining the sorts and function symbols for the cryptographic functions and for all the state constructor symbols, and $E_{\mathcal{P}}$ is a set of equations specifying the *algebraic properties* of the cryptographic functions and the state constructors. A protocol $\mathcal{P}$ is specified with a notation derived from strand spaces [13]. In a *strand*, a local execution of a protocol by a principal is indicated by a sequence of messages $[msg_1^-,\ msg_2^+,\ msg_3^-, \ldots,\ msg_{k-1}^-,\ msg_k^+]$ where each msg_i is a term of sort Msg (i.e., $msg_i \in T_{\Sigma_{\mathcal{P}}/E_{\mathcal{P}}}(X)_{\mathsf{Msg}}$). Strand items representing input messages are assigned a negative sign, and strand items representing output messages are assigned a positive sign. For each positive message msg_i in a sequence of messages $[msg_1^{\pm},\ msg_2^{\pm},\ msg_3^{\pm}, \ldots,\ msg_i^{+}, \ldots,\ msg_{k-1}^{\pm},\ msg_k^{\pm}]$ the non-fresh variables (see below) occurring in an output message msg_i^+ must appear in previous messages $msg_1, msg_2, msg_3, \ldots, msg_{i-1}$. In Maude-NPA [10,11], strands evolve over time and thus we use the symbol $|$ to divide past and future in a strand, i.e., $[nil, msg_1^{\pm}, \ldots, msg_{j-1}^{\pm} \mid msg_j^{\pm}, msg_{j+1}^{\pm}, \ldots, msg_k^{\pm}, nil]$, where $msg_1^{\pm}, \ldots,$

$msg^{\pm}_{j-1}$ are the past messages, and $msg^{\pm}_{j}, msg^{\pm}_{j+1}, \ldots, msg^{\pm}_{k}$ are the future messages ($msg^{\pm}_{j}$ is the immediate future message). The nils are present so that the bar may be placed at the beginning or end of the strand if necessary. A strand $[msg^{\pm}_{1}, \ldots, msg^{\pm}_{k}]$ is a shorthand for $[nil \mid msg^{\pm}_{1}, \ldots, msg^{\pm}_{k}, nil]$. We often remove the nils for clarity, except when there is nothing else between the vertical bar and the beginning or end of a strand. We write $\mathcal{P}$ for the set of strands in a protocol, including the strands that describe the intruder's behavior.

Maude-NPA uses a special sort Msg of messages that allows the protocol specifier to describe other sorts as subsorts of the top sort Msg. The specifier can make use of a special sort Fresh in the protocol-specific signature Σ for representing fresh unguessable values, e.g., nonces. The meaning of a variable of sort Fresh is that it will never be instantiated by an E-unifier generated during the backwards reachability analysis. This ensures that if two nonces are represented using different variables of sort Fresh, they will never be identified and no approximation for nonces is necessary. We make explicit the Fresh variables $r_1, \ldots, r_k (k \geq 0)$ generated by a strand by writing $:: r_1, \ldots, r_k :: [msg^{\pm}_{1}, \ldots, msg^{\pm}_{n}]$, where $r_1, \ldots, r_k$ appear somewhere in $msg^{\pm}_{1}, \ldots, msg^{\pm}_{n}$. Fresh variables generated by a strand are unique to that strand.

A *state* is a set of Maude-NPA strands unioned together by an associative and commutativity union operator _&_ with identity operator $\emptyset$, along with an additional term describing the intruder knowledge at that point. The *intruder knowledge* is represented as a set of facts unioned together with an associative and commutativity union operator _,_ with identity operator $\emptyset$. There are two kinds of intruder facts: positive knowledge facts (the intruder knows message m, i.e., $m \in \mathcal{I}$), and negative knowledge facts (the intruder does not yet know m but will know it in a future state, i.e., $m \notin \mathcal{I}$).

In the case in which new strands are not introduced into the state, the rewrite rules $R_{\mathcal{P}}$ obtained from the protocol strands $\mathcal{P}$ are as follows[2], where L, L_1, L_2 denote lists of input and output messages ($+m$,$-m$), IK, IK' denote sets of intruder facts ($m \in \mathcal{I}$,$m \notin \mathcal{I}$), and SS, SS' denote sets of strands:

$$SS \ \& \ [L \mid M^-, L'] \ \& \ (M \in \mathcal{I}, IK) \to SS \ \& \ [L, M^- \mid L'] \ \& \ (M \in \mathcal{I}, IK) \quad (1)$$

$$SS \ \& \ [L \mid M^+, L'] \ \& \ IK \to SS \ \& \ [L, M^+ \mid L'] \ \& \ IK \quad (2)$$

$$SS \ \& \ [L \mid M^+, L'] \ \& \ (M \notin \mathcal{I}, IK) \to SS \ \& \ [L, M^+ \mid L'] \ \& \ (M \in \mathcal{I}, IK) \quad (3)$$

In a *forward execution* of the protocol strands, Rule (1) synchronizes an input message with a message already in the channel (i.e., learned by the intruder), Rule (2) accepts output messages but the intruder's knowledge is not increased, and Rule (3) accepts output messages and the intruder's knowledge is positively increased. Note that Rule (3) makes explicit *when* the intruder learned a message M, which is recorded in the previous state by the negative fact $M \notin \mathcal{I}$. A fact $M \notin \mathcal{I}$ can be paraphrased as: "the intruder does not yet know M, but will learn it in the future".

[2] Note that to simplify the exposition, we omit the fresh variables at the beginning of each strand in a rewrite rule.

New strands are added to the state by explicit introduction through dedicated rewrite rules (one for each honest or intruder strand). It is also the case that when we are performing a backwards search, only the strands that we are searching for are listed explicitly, and extra strands necessary to reach an initial state are dynamically added. Thus, when we want to introduce new strands into the explicit description of the state, we need to describe additional rules for doing that, as follows:

$$\text{for each } [\ l_1,\ u^+,\ l_2\ ] \in \mathcal{P} : SS\ \ \&\,[\ l_1 \mid u^+, l_2\ ]\,\&\,(u{\notin}\mathcal{I}, IK) \to SS\ \ \&\,(u{\in}\mathcal{I}, IK) \quad (4)$$

where u denotes a message, l_1, l_2 denote lists of input and output messages ($+m$,$-m$), IK denotes a set of intruder facts ($m{\in}\mathcal{I}$,$m{\notin}\mathcal{I}$), and SS denotes a set of strands. For example, intruder concatenation of two learned messages is described as follows:

$$SS\ \&\ [M_1^-, M_2^- \mid (M_1; M_2)^+]\ \&\ ((M_1;M_2){\notin}\mathcal{I}, IK) \to SS\ \&\ ((M_1;M_2){\in}\mathcal{I}, IK)$$

In summary, for a protocol $\mathcal{P}$, the set of rewrite rules obtained from the protocol strands that are used for backwards narrowing reachability analysis *modulo* the equational properties $E_{\mathcal{P}}$ is $R_{\mathcal{P}} = \{(1),(2),(3)\} \cup (4)$.

5 Syntax for Protocol Specification and Composition

We begin by describing the new syntactic features we need to make explicit in each protocol to later define sequential protocol compositions. Each strand in a protocol specification in the Maude-NPA is now extended with *input parameters* and *output parameters*. Input parameters are a sequence of variables of different sorts placed at the beginning of a strand. Output parameters are a sequence of terms placed at the end of a strand. Any variable contained in an output parameter must appear either in the body of the strand, or as an input parameter. The strand notation we will now use is $[\{\overrightarrow{I}\}, \overrightarrow{M}, \{\overrightarrow{O}\}]$ where $\overrightarrow{I}$ is a list of input parameter variables, $\overrightarrow{M}$ is a list of positive and negative terms in the strand notation of the Maude-NPA, and $\overrightarrow{O}$ is a list of output terms all of whose variables appear in $\overrightarrow{M}$ or $\overrightarrow{I}$. The input and output parameters describe the exact assumptions about each principal.

In the following, we first describe our syntax for protocol specification and then introduce a new syntax for protocol composition. Similarly to the Maude syntax for modules, we define a protocol modularly as follows:

```
prot Name is sorts Sorts . subsorts Subsorts .
    Operators Variables Equations DYStrands Strands
endp
```

where *Name* is a valid Maude module name, *Sorts* is a valid Maude-NPA declaration of sorts, *Subsorts* is a valid Maude-NPA declaration of subsorts, *Operators* is a valid Maude-NPA declaration of operators, *Variables* is a valid Maude-NPA declaration of variables to be used in the equational properties and in the honest and Dolev-Yao strands, *Equations* is a valid Maude-NPA declaration of equational properties, *DYStrands* is a sequence of valid Maude-NPA Dolev-Yao strands, each starting with the word `DYstrand` and ending with a period, and *Strands* is a sequence of valid Maude-NPA strands, each starting with the word `strand` and ending with a period. The Maude-NPA protocol specifications of all the examples can be found in `http://www.dsic.upv.es/~ssantiago/composition.html`.

Example 1. The following description of the NSL protocol contains more technical details than the informal description of NSL in Section 2. A nonce generated by principal A is denoted by $n(A, r)$, where r is a unique variable of sort Fresh. Concatenation of two messages, e.g., N_A and N_B, is denoted by the operator $\_;\_$, e.g., $n(A,r)\ ;\ n(B,r')$. Encryption of a message M with the public key K_A of principal A is denoted by $pk(A,M)$, e.g., $\{N_B\}_{pk(B)}$ is denoted by $pk(B, n(B, r'))$. Encryption with a secret key is denoted by $sk(A,M)$. The public/private encryption cancellation properties are described using the equations $pk(X, sk(X, Z)) = Z$ and $sk(X, pk(X, Z)) = Z$. The two strands $\mathcal{P}$ associated to the three protocol steps shown above are as follows:

```
strand [NSL.init] :: r :: [ {A,B} |
 +(pk(B,n(A,r);A)), -(pk(A,n(A,r);N;B)), +(pk(B,N)), {A,B,n(A,r),N}] .
strand [NSL.resp] :: r :: [ {A,B} |
 -(pk(B,N;A)), +(pk(A,N;n(B,r);B)), -(pk(B,n(B,r))), {A,B,N,n(B,r)}] .
```

Note that we allow each honest or Dolev-Yao strand to be *labeled* (e.g. `init` or `resp`), in contrast to the standard Maude-NPA syntax for strands. These strand labels play an important role in our protocol composition method as explained below.

Example 2. Similarly to the NSL protocol, there are several technical details missing in the previous informal description of DB. The exclusive-or operator is $\_\oplus\_$ and its equational properties are described using associativity and commutativity of $\oplus$ plus the equations[3] $X \oplus 0 = X$, $X \oplus X = 0$, and $X \oplus X \oplus Y = Y$. Since Maude-NPA does not yet include timestamps, we do not include all the actions relevant to calculating time intervals, sending timestamps, and checking them. The two strands $\mathcal{P}$ associated to the three protocol steps shown above are as follows:

```
strand [DB.init] :: r :: [{A,B,NA} |
 +(n(B,r)), -(n(B,r)*NA),  {A,B,NA,n(B,r)}] .
strand [DB.resp] :: nil :: [{A,B,NA} | -(NB), +(NB * NA), {A,B,NA,NB}] .
```

In this protocol specification, it is made clear that the nonce N_A used by the initiator is a parameter and is never generated by A during the run of DB. However, the initiator B does generate a new nonce.

[3] Note that the redundant equational property $X \oplus X \oplus Y = Y$ is necessary in Maude-NPA for coherence purposes; see [11].

Example 3. The previous informal description of the KD protocol also lacks several technical details, which we supply here. Encryption of a message M with key K is denoted by $e(K, M)$, e.g., $\{N'_B\}_{h(N_A,N_B)}$ is denoted by $e(h(n(A, r), n(B, r')),$ $n(B, r''))$. Cancellation properties of encryption and decryption are described using the equations $e(X, d(X, Z)) = Z$ and $d(X, e(X, Z)) = Z$. Session keys are written $skey(A, r)$, where A is the principal's name and r is a Fresh variable. The two strands $\mathcal{P}$ associated to the KD protocol steps shown above are as follows:

```
strand [KD.init] :: r :: [ {A,B,K} | +(e(K,skey(A,r)),
  -(e(K,skey(A,r) ; N)), +(e(K, N)), {A,B,K,skey(A,r),N}] .
strand [KD.resp] :: r :: [ {A,B,K} | -(e(K,SK)), +(e(K,SK ; n(B,r))),
  -(e(K,n(B,r))), {A,B,K,SK,n(B,r)} ] .
```

Sequential composition of two strands describes a situation in which one strand (the child strand), can only execute after the parent strand has completed execution. Each composition of two strands is obtained by matching the output parameters of the parent strand with the input parameters of the child strand in a user-defined way. Note that it may be possible for a single parent strand to have more than one child strand.

The relevant fact in the DB protocol is that both nonces are required to be unknown to an attacker before they are sent, but the nonce originating from the responder must be previously agreed upon between the two principals. Therefore, this protocol is usually composed with another protocol ensuring secrecy and authentication of nonces. Furthermore, according to [16], there are two extra issues related to the DB protocol that must be considered: (i) the initiator of the previous protocol plays the role of the responder in DB and viceversa, and (ii) nonces generated by the parent protocol cannot be shared by more than one child so that an initiator of NSL will be connected to one and only one responder of DB. In our working example, we use the NSL protocol to provide these capabilities.

Similarly to the syntax for protocols, we define protocol composition as follows[4]:

prot *Name* **is** *Name1* ; *Name2*
  $a_1\{\overrightarrow{O_1}\}$; $\{\overrightarrow{I_1}\}b_1$ [1-1] (or [1-*]) . $\cdots$ $a_n\{\overrightarrow{O_n}\}$; $\{\overrightarrow{I_n}\}b_n$ [1-1] (or [1-*]) .
endp

where *Name* is a valid Maude-NPA module name, *Name1* and *Name2* are protocol names previously defined, $a_1, \ldots, a_n$ are labels of strands in protocol *Name1*, and $b_1, \ldots, b_n$ are labels of strands in protocol *Name2*. Furthermore, for each composition $a_i\{\overrightarrow{O_i}\};\{\overrightarrow{I_i}\}b_i$, strand definition $:: \overrightarrow{r_{a_i}} :: [\{\overrightarrow{I_{a_i}}\}, \overrightarrow{a_i}, \{\overrightarrow{O_{a_i}}\}]$ for role a_i, and strand definition $:: \overrightarrow{r_{b_i}} :: [\{\overrightarrow{I_{b_i}}\}, \overrightarrow{b_i}, \{\overrightarrow{O_{b_i}}\}]$ for role b_i, we have that:

1. variables are properly renamed, i.e. $V_{ab} = \mathcal{V}ar(\overrightarrow{I_i}) \cup \mathcal{V}ar(\overrightarrow{O_i})$, $V_a = \mathcal{V}ar(\overrightarrow{I_{a_i}}) \cup$ $\mathcal{V}ar(\overrightarrow{O_{a_i}})$, $V_b = \mathcal{V}ar(\overrightarrow{I_{b_i}}) \cup \mathcal{V}ar(\overrightarrow{O_{b_i}})$, and $V_{ab} \cap V_a \cap V_b = \emptyset$;
2. the variables of $\overrightarrow{I_i}$ must appear in $\overrightarrow{O_i}$ (no extra variables are allowed in a protocol composition);

[4] Operator and sort renaming is indeed necessary, as in the Maude module importation language, but we do not consider those details in this paper.

3. the formal output parameters $\overrightarrow{O_{a_i}}$ must match the actual output parameters $\overrightarrow{O_i}$, i.e., $\exists \sigma_a$ s.t. $\overrightarrow{O_{a_i}} =_{E_{\mathcal{P}}} \sigma_a(\overrightarrow{O_i})$; and
4. the actual input parameters $\overrightarrow{I_i}$ must match the formal input parameters $\overrightarrow{I_{b_i}}$, i.e., $\exists \sigma_b$ s.t. $\overrightarrow{I_i} =_{E_{\mathcal{P}}} \sigma_b(\overrightarrow{I_{b_i}})$.

The expressions $[1{-}1]$ (or $[1{-}*]$) indicate whether a one-to-one (or a one-to-many) composition is desired for those two strands. Note that for each composition, if there are substitutions σ_a and σ_b as described above, then there is a substitution σ_{ab} combining both, i.e., $\sigma_{ab}(X) = \sigma_a(\sigma_b(X))$ for any variable X, and then $\sigma_a(\overrightarrow{I_i}) =_{E_{\mathcal{P}}} \sigma_{ab}(\overrightarrow{I_{b_i}})$. This ensures that any protocol composition is feasible and avoids the possibility of failing protocol compositions.

Let us consider again our two NSL and DB protocols and their composition. Note that we do not have to modify either the NSL or the DB specification above. The composition of both protocols is specified as follows:

```
prot NSL-DB is NSL ; DB
  NSL.init {A,B,NA,NB} ; {A,B,NA} DB.resp [1-1] .
  NSL.resp {A,B,NA,NB} ; {A,B,NA} DB.init [1-1] .
endp
```

Let us now consider the NSL and KD protocols and their composition. The composition of both protocols, which is an example of a one-to-many composition, is specified as follows:

```
prot NSL-KD is NSL ; KB
  NSL.init {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.resp [1-*] .
  NSL.init {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.init [1-*] .
  NSL.resp {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.init [1-*] .
  NSL.resp {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.resp [1-*] .
endp
```

In the remainder of this paper we remove irrelevant parameters (i.e. input parameters for strands with no parents, and output parameters for strands with no children) in order to simplify the exposition.

6 Maude-NPA's Composition Execution Model

In this section we define a concrete execution model for the one-to-one and one-to-many protocol compositions by extending the Maude-NPA execution model. However, we show that, by a simple protocol transformation, we are able to analyze and verify this dynamic composition in the current Maude-NPA tool. We prove soundness and completeness of the protocol transformation with respect to the extended operational semantics, and illustrate our results on our two running examples.

6.1 Composition Execution Model

As explained in Section 4, the operational semantics of protocol execution and analysis is based on rewrite rules denoting state transitions which are applied

for each one-to-one composition $\{a\{\overrightarrow{O}\}; \{\overrightarrow{I}\}b\}$ [1−1] with
strand definition $[\{\overrightarrow{I_a}\}, \overrightarrow{a}, \{\overrightarrow{O_a}\}]$ for protocol a,
strand definition $[\{\overrightarrow{I_b}\}, \overrightarrow{b}, \{\overrightarrow{O_b}\}]$ for protocol b,
and substitutions σ_a, σ_{ab} s.t. $\overrightarrow{O_a} =_{E_{\mathcal{P}}} \sigma_a(\overrightarrow{O})$ and $\sigma_a(\overrightarrow{I}) =_{E_{\mathcal{P}}} \sigma_{ab}(\overrightarrow{I_b})$,
we add the following rule :

$$\begin{aligned} & SS \,\&\, \overrightarrow{a} \mid \{\overrightarrow{O_a}\}] \;\&\; [nil \mid \{\sigma_{ab}(\overrightarrow{I_b})\}, \sigma_{ab}(\overrightarrow{b})] \,\&\, IK \\ \to\ & SS \,\&\, \overrightarrow{a}, \{\overrightarrow{O_a}\} \mid nil] \;\&\; [\{\sigma_{ab}(\overrightarrow{I_b})\} \mid \sigma_{ab}(\overrightarrow{b})] \,\&\, IK \end{aligned} \tag{5}$$

$$\begin{aligned} & SS \,\&\, \overrightarrow{a} \mid \{\overrightarrow{O}\}] \;\&\; [nil \mid \{\sigma_{ab}(\overrightarrow{I_b})\}, \sigma_{ab}(\overrightarrow{b})] \,\&\, IK \\ \to\ & SS \,\&\, [\{\sigma_{ab}(\overrightarrow{I_b})\} \mid \sigma_{ab}(\overrightarrow{b})] \,\&\, IK \end{aligned} \tag{6}$$

Fig. 1. Semantics for one-to-one composition

modulo the algebraic properties $E_{\mathcal{P}}$ of the given protocol $\mathcal{P}$. Therefore, in the one-to-one and one-to-many cases we must add new state transition rules in order to deal with protocol composition. Maude-NPA performs backwards search modulo $E_{\mathcal{P}}$ by reversing the transition rules expressed in a forward way; see [10,11]. Again, we define forward rewrite rules which will happen to be executed in a backwards way.[5]

In the one-to-one composition, we add the state transition rules of Figure 1. Rule 5 composes a parent and a child strand already present in the current state. Rule 6 adds a parent strand to the current state and composes it with an existing child strand. Note that since a strand specification is a symbolic specification representing many concrete instances and the same applies to a composition of two protocol specifications, we need to relate actual and formal parameters of the protocol composition w.r.t. the two protocol specifications by using the substitutions σ_a and σ_{ab} in Figure 1. For example, given the following composition of the NSL-DB protocol

```
  NSL.init {A,B,NA,NB} ; {A,B,NA} DB.resp [1-1] .
```

where NSL.init and DB.resp were defined in Section 5, we add the following transition rule for Rule (5) where both the parent and the child strands are present and thus synchronized

```
:: r :: [ +(pk(B,n(A,r);A)), -(pk(A,n(A,r);N;B)), +(pk(B,N))
        | {A,B,n(A,r),N} ]
:: nil :: [ nil | {A,B,n(A,r)}, -(NB), +(NB * n(A,r)) ] & SS & IK
->
:: r :: [ +(pk(B,n(A,r);A)), -(pk(A,n(A,r);N;B)), +(pk(B,N)),
```

[5] Note however that we represent unification explicitly via a substitution σ instead of implicitly via variable equality as in Section 4. This is because output and input parameters are not required to match, e.g. in the composition NSL-KD, the output parameters of the parent strand are $\{A, B, N_A, N_B\}$ whereas the input parameters of the child strand are $\{A, B, h(N_A, N_B)\}$.

for each one-to-many composition $\{a\{\overrightarrow{O}\}; \{\overrightarrow{I}\}b\}$ [1−∗] with
strand definition $[\{\overrightarrow{I_a}\}, \overrightarrow{a}, \{\overrightarrow{O_a}\}]$ for protocol a,
strand definition $[\{\overrightarrow{I_b}\}, \overrightarrow{b}, \{\overrightarrow{O_b}\}]$ for protocol b,
and substitutions σ_a, σ_{ab} s.t. $\overrightarrow{O_a} =_{E_{\mathcal{P}}} \sigma_a(\overrightarrow{O})$ and $\sigma_a(\overrightarrow{I}) =_{E_{\mathcal{P}}} \sigma_{ab}(\overrightarrow{I_b})$,
we add one Rule 5, one Rule 6, and the following rule :

$$\begin{aligned} & SS \,\&\, [\overrightarrow{a} \mid \{\overrightarrow{O_a}\}] \;\&\; [nil \mid \{\sigma_{ab}(\overrightarrow{I_b})\}, \sigma_{ab}(\overrightarrow{b})] \,\&\, IK \\ \rightarrow\; & SS \,\&\, [\overrightarrow{a} \mid \{\overrightarrow{O_a}\}] \;\&\; [\{\sigma_{ab}(\overrightarrow{I_b})\} \mid \sigma_{ab}(\overrightarrow{b})] \,\&\, IK \end{aligned} \tag{7}$$

Fig. 2. Semantics for one-to-many composition

```
          {A,B,n(A,r),N} | nil ]
:: nil :: [{A,B,n(A,r)} | -(NB), +(NB * n(A,r)) ] & SS & IK
```

One-to-many composition uses the rules in Figure 1 for the first child plus an additional rule for subsequent children, described in Figure 2. Rule 7 composes a parent strand and a child strand but the bar in the parent strand is not moved, in order to allow further backwards child compositions. For example, given the following composition of the NSL-KD protocol

```
  NSL.resp {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.init [1-*] .
```

where NSL.resp and KD.init are as defined in Section 5, we add the following transition rule for Rule (7) using substitution σ_{ab} = {`A'` ↦ `A`, `B'` ↦ `B`, `K` ↦ `h(NA,n(B,r))`}:

```
:: r :: [ -(pk(B,NA;A)), +(pk(A,NA;n(B,r);B)), -(pk(B,n(B,r)))
          | {A,B,NA,n(B,r)}] .
:: r' :: [ nil | {A,B,h(NA,n(B,r))}, +(e(h(NA,n(B,r)),skey(A,r')),
           -(e(h(NA,n(B,r)),skey(A,r') ; N)), +(e(h(NA,n(B,r)), N)) ] .
& SS & IK
->
:: r :: [ -(pk(B,NA;A)), +(pk(A,NA;n(B,r);B)), -(pk(B,n(B,r))),
          {A,B,NA,n(B,r)} | nil ] .
:: r' :: [ {A,B,h(NA,n(B,r))} | +(e(h(NA,n(B,r)),skey(A,r')),
           -(e(h(NA,n(B,r)),skey(A,r') ; N)), +(e(h(NA,n(B,r)), N)) ] .
& SS & IK
```

Thus, for a protocol composition $\mathcal{P}_1;\mathcal{P}_2$, the rewrite rules governing protocol execution are $R^{\circ}_{\mathcal{P}_1;\mathcal{P}_2} = \{(1),(2),(3)\} \cup (4) \cup (5) \cup (6) \cup (7)$.

6.2 Protocol Composition by Protocol Transformation

Instead of implementing a new version of the Maude-NPA generating new transition rules for each protocol composition, we have defined a *protocol transformation* that achieves the same effect using the current Maude-NPA tool.

The protocol transformation is given in Figure 3. Its output is a single, composed protocol specification where:

$$\Phi(\mathcal{P}_1;\mathcal{P}_2) = \begin{cases} \text{add strand } [\overrightarrow{a}, -(role_b(r)), +(role_a(r) \;.\; \sigma_{ab}(\dot{I}_b))] \text{ and} \\ \text{strand } [+(role_b(r)), -(role_a(r) \;.\; \dot{I}_b), \overrightarrow{b}] \\ \qquad \text{whenever } \{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\ [1{-}1] \text{ in } \mathcal{P}_1;\mathcal{P}_2, \\ \qquad\qquad \text{strand definition } [role_a][\{\overrightarrow{I_a}\}, \overrightarrow{a}, \{\overrightarrow{O_a}\}] \text{ for protocol } a, \\ \qquad\qquad \text{strand definition } [role_b][\{\overrightarrow{I_b}\}, \overrightarrow{b}, \{\overrightarrow{O_b}\}] \text{ for protocol } b, \\ \qquad\qquad \exists \sigma_a, \sigma_{ab} \text{ s.t. } \overrightarrow{O_a} =_{E_{\mathcal{P}}} \sigma_a(\overrightarrow{O}) \text{ and } \sigma_a(\overrightarrow{I}) =_{E_{\mathcal{P}}} \sigma_{ab}(\overrightarrow{I_b}) \\ \qquad\qquad \text{and } r \text{ is a fresh variable} \\ \text{add strand } [\overrightarrow{a}, +(role_a(r) \;.\; \sigma_{ab}(\dot{I}_b))] \text{ and strand } [-(role_a(r) \;.\; \dot{I}_b), \overrightarrow{b}] \\ \qquad \text{whenever } \{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\ [1{-}*] \text{ in } \mathcal{P}_1;\mathcal{P}_2, \\ \qquad\qquad \text{strand definition } [role_a][\{\overrightarrow{I_a}\}, \overrightarrow{a}, \{\overrightarrow{O_a}\}] \text{ for protocol } a, \\ \qquad\qquad \text{strand definition } [role_b][\{\overrightarrow{I_b}\}, \overrightarrow{b}, \{\overrightarrow{O_b}\}] \text{ for protocol } b, \\ \qquad\qquad \exists \sigma_a, \sigma_{ab} \text{ s.t. } \overrightarrow{O_a} =_{E_{\mathcal{P}}} \sigma_a(\overrightarrow{O}) \text{ and } \sigma_a(\overrightarrow{I}) =_{E_{\mathcal{P}}} \sigma_{ab}(\overrightarrow{I_b}) \\ \qquad\qquad \text{and } r \text{ is a fresh variable} \end{cases}$$

Fig. 3. Protocol Transformation

1. Sorts, symbols, and equational properties of both protocols are put together into a single specification[6]. Strands of both protocols are transformed and added to this single specification as described in Figure 3.
2. For each composition we transform the input parameters $\{\overrightarrow{I_b}\}$ into an input message exchange of the form $-(\overrightarrow{I_b})$, and the output parameters $\{\overrightarrow{O_a}\}$ into an output message exchange of the form $+(\sigma_{ab}(\overrightarrow{I_b}))$. The sort Param of these messages is disjoint from the sort Msg used by the protocol in the honest and intruder strands. This ensures that they are harmless, since no intruder strand will be able to use them. In order to avoid type conflicts, we use a *dot* for concatenation within protocol composition exchange messages, e.g. input parameters $\overrightarrow{I} = \{A, B, NA\}$ are transformed into the sequence $\dot{I} = A \;.\; B \;.\; NA$.
3. Each composition is uniquely identified by using a composition identifier (a variable of sort Fresh). Strands exchange such composition identifier by using input/output messages of the form $role_j(r)$, which make the role explicit. The sort Role of these messages is disjoint from the sorts Param and Msg.
   (a) In a one-to-one protocol composition, the child strand uniquely generates a fresh variable that is added to the area of fresh identifiers at the beginning of its strand specification. This fresh variable must be passed from the child to the parent before the parent generates its output parameters and sends it back again to the child.
   (b) In a one-to-many protocol composition, the parent strand uniquely generates a fresh variable that is passed to the child. Since an (a priori) unbounded number of children will be composed with it, no reply of the fresh variable is expected by the parent from the children.

[6] Note that we allow shared items but require the user to solve any possible conflict. Operator and sort renaming is an option, as in the Maude module importation language, but we do not consider those details in this paper.

For example, for the following one-to-one protocol composition in NSL-DB

```
NSL.init {A,B,NA,NB} ; {A,B,NA} DB.resp [1-1] .
```

where NSL.init and DB.resp are as defined in Section 5 we have the following two transformed strands:

```
[NSL.init] :: r :: [ +(pk(B,n(A,r);A)), -(pk(A,n(A,r);N;B)), +(pk(B,N)),
                     -(db-resp(r#)), +(nsl-init(r#) . A . B . n(A,r)) ] .
[DB.resp] :: r# :: [ +(db-resp(r#)), -(nsl-init(r#) . A . B . NA),
                     -(NB), +(NB * NA) ] .
```

For the following one-to-many protocol composition in the NSL-KD

```
NSL.resp {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.init [1-*] .
```

where NSL.resp and KD.init are as defined in Section 5 we have the following two transformed strands:

```
[NSL.resp] :: r,r# :: [-(pk(B,N;A)), +(pk(A,N;n(B,r);B)), -(pk(B,n(B,r))),
                        +(nsl-init(r#) . A . B . h(N,n(B,r))) ] .
[KD.init] :: r :: [ -(nsl-init(r#) . A . B . K), +(e(K,skey(A,r)),
                    -(e(K,skey(A,r) ; N')), +(e(K,N')) ] .
```

Soundness and completeness for this protocol transformation is provided in the following result. The proof of this theorem is available in [12]. Note that a state *St* is called *associated to a rewrite theory* $\mathcal{R}$ if it is a valid term w.r.t. the order-sorted signature of $\mathcal{R}$. We call a state *initial* if there are no backwards narrowing steps from it.

Theorem 1 (Soundness and Completeness). *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1;\mathcal{P}_2$ their composition, as defined in Section 5. Let $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ be the composition rewrite theory defined in Section 6.2, and let $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ be the original Maude-NPA rewrite theory, as described in Section 6.1. Then there is a binary relation $\equiv_{\Phi}$ between the states associated to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ and the states associated to the rewrite theory $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ such that given St and St' associated to $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ and $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$, respectively, then $St \equiv_{\Phi} St'$ implies that there is an initial state In reachable from St by backwards narrowing in $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ iff there is an initial state In' such that $In \equiv_{\Phi} In'$ and In' is reachable from St' by backwards narrowing in $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$.*

We also have experimental results for protocol composition based on this theorem: (i) we found the attack for the NSL-DB protocol composition, (ii) we proved the security of the fixed version of the NSL-DB composition using a hash function, and (iii) we proved the security of NSL-KD. Due to space limitations, our experiments are not included here but they are available at `http://www.dsic.upv.es/~ssantiago/composition.html`.

7 Related Work and Conclusions

Our work addresses a somewhat different problem than most existing work on cryptographic protocol composition, which generally does not address model-checking. Indeed, to the best of our knowledge, most protocol analysis model-checking tools simply use hand-coded concatenation of protocol specifications to express sequential composition. However, we believe that the problem we are addressing is an important one that tackles a widely acknowledged source of protocol complexity. For example, in the Internet Key Exchange Protocol [18] there are sixteen different one-to-many parent-child compositions of Phase One and Phase Two protocols. The ability to synthesize compositions automatically would greatly simplify the specification and analysis of protocols like these.

Now that we have a mechanism for synthesizing compositions, we are ready to revisit existing research on composing protocols and their properties and determine how we could best make use of it in our framework. There have been two approaches to this problem. One (called *nondestructive* composition in [6]) is to concentrate on properties of protocols and conditions on them that guarantee that properties satisfied separately are not violated by the composition. This is, for example, the approach taken by Gong and Syverson [14], Guttman and Thayer [17], Cortier and Delaune [5] and, in the computational model, Canetti's Universal Composability [2]. The conditions in this case are usually ones that can be verified syntactically, so Maude-NPA, or any other model checker, would not be of much assistance here.

Of more interest to us is the research that addresses the compositionality of the protocol properties themselves (called *additive* composition in [6]). This addresses the development of logical systems and tools such as CPL, PDL, and CPSA cited earlier in this paper, in which inference rules are provided for deriving complex properties of a protocol from simpler ones. Since these are pure logical systems, they necessarily start from very basic statements concerning, for example, what a principal can derive when it receives a message. But there is no reason why the properties of the component protocols could not be derived using model checking, and then composed using the logic. This would give us the benefits of both model checking (for finding errors and debugging), and logical derivations (for building complex systems out of simple components), allowing to switch between one and the other as needed. Indeed, Maude-NPA is well positioned in that respect. For example, the notion of state in strand spaces that it uses is very similar to that used by PDL [4], and we have already developed a simple property language that allows us to translate the "shapes" produced by CPSA into Maude-NPA attack states. The next step in our research will be to investigate the connection more closely from the point of view of compositionality.

References

1. Anlauff, M., Pavlovic, D., Waldinger, R., Westfold, S.: Proving authentication properties in the protocol derivation assistant. In: Proc. of Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (2006)
2. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC, pp. 494–503 (2002)
3. Capkun, S., Hubaux, J.P.: Secure positioning in wireless networks. IEEE Journal on Selected Areas in Communication 24(2) (February 2006)
4. Cervesato, I., Meadows, C., Pavlovic, D.: An encapsulated authentication logic for reasoning about key establishment protocols. In: IEEE Computer Security Foundations Workshop (2005)

5. Cortier, V., Delaune, S.: Safely composing security protocols. Formal Methods in System Design 34(1), 1–36 (2009)
6. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: Secure protocol composition. In: Proc. Mathematical Foundations of Programming Semantics. ENTCS, vol. 83 (2003)
7. Desmedt, Y.: Major security problems with the "unforgeable" (Feige-)Fiat-Shamir Proofs of identity and how to overcome them. In: Securicom 88, 6th Worldwide Congress on Computer and Communications Security and Protection, Paris, France, March 1988, pp. 147–159 (1988)
8. Doghim, S., Guttman, J., Thayer, F.J.: Searching for Shapes in Cryptographic Protocols. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 523–537. Springer, Heidelberg (2007)
9. Durgin, N., Mitchell, J., Pavlovic, D.: A Compositional Logic for Program Correctness. In: Fifteenth Computer Security Foundations Workshop — CSFW-14, Cape Breton, NS, Canada, June 11-13, IEEE Computer Society Press, Los Alamitos (2001)
10. Escobar, S., Meadows, C., Meseguer, J.: A rewriting-based inference system for the NRL Protocol Analyzer and its meta-logical properties. Theor. Comput. Sci. 367(1-2), 162–202 (2006)
11. Escobar, S., Meadows, C., Meseguer, J.: Maude-NPA: Cryptographic protocol analysis modulo equational properties. In: FOSAD 2008/2009 Tutorial Lectures, vol. 5705, pp. 1–50. Springer, Heidelberg (2009)
12. Escobar, S., Meadows, C., Meseguer, J., Santiago, S.: Sequential Protocol Composition in Maude-NPA. Technical Report DSIC-II/06/10, Universidad Politécnica de Valencia (June 2010)
13. Thayer Fabrega, F.J., Herzog, J., Guttman, J.: Strand Spaces: What Makes a Security Protocol Correct? Journal of Computer Security 7, 191–230 (1999)
14. Gong, L., Syverson, P.: Fail-stop protocols: An approach to designing secure protocols. In: Proc. of the 5th IFIP International Working Conference on Dependable Computing for Critical Applications, pp. 79–99. IEEE Computer Society Press, Los Alamitos (1998)
15. Guttman, J.: Security protocol design via authentication tests. In: Proc. Computer Security Foundations Workshop. IEEE Computer Society Press, Los Alamitos (2001)
16. Guttman, J.D., Herzog, J.C., Swarup, V., Thayer, F.J.: Strand spaces: From key exchange to secure location. In: Workshop on Event-Based Semantics (2008)
17. Guttman, J.D., Thayer, F.J.: Protocol independence through disjoint encryption. In: CSFW, pp. 24–34 (2000)
18. Harkins, D., Carrel, D.: The Internet Key Exchange (IKE), IETF RFC 2409 (November 1998)
19. Lowe, G.: Breaking and fixing the Needham-Schroeder public key protocol using FDR. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)
20. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theor. Comput. Sci. 96(1), 73–155 (1992)
21. Meseguer, J.: Membership algebra as a logical framework for equational specification. In: Parisi-Presicce, F. (ed.) WADT 1997. LNCS, vol. 1376, pp. 18–61. Springer, Heidelberg (1998)
22. TeReSe (ed.): Term Rewriting Systems. Cambridge University Press, Cambridge (2003)